Maya+Pythonでインハウスツールを作ろう！

ポリフォニー・デジタル

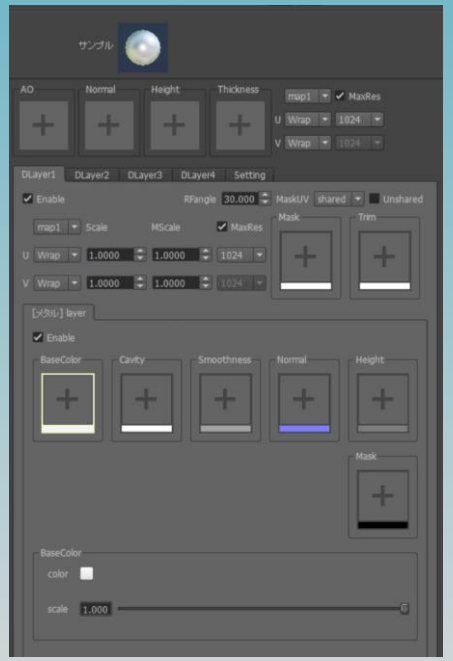
富田岳伸

本日のアジェンダ

- ポリフォニー・デジタルでのPython活用事例
- アセット管理ツールを作ってみよう
- Pythonコードの高速化Tips

ポリフォニー・デジタルでのPython活用事例

- マテリアルワークフロー
  - MayaのアトリビュートエディターをPySideで書き換え
  - GTのマテリアルワークフローに特化したUI

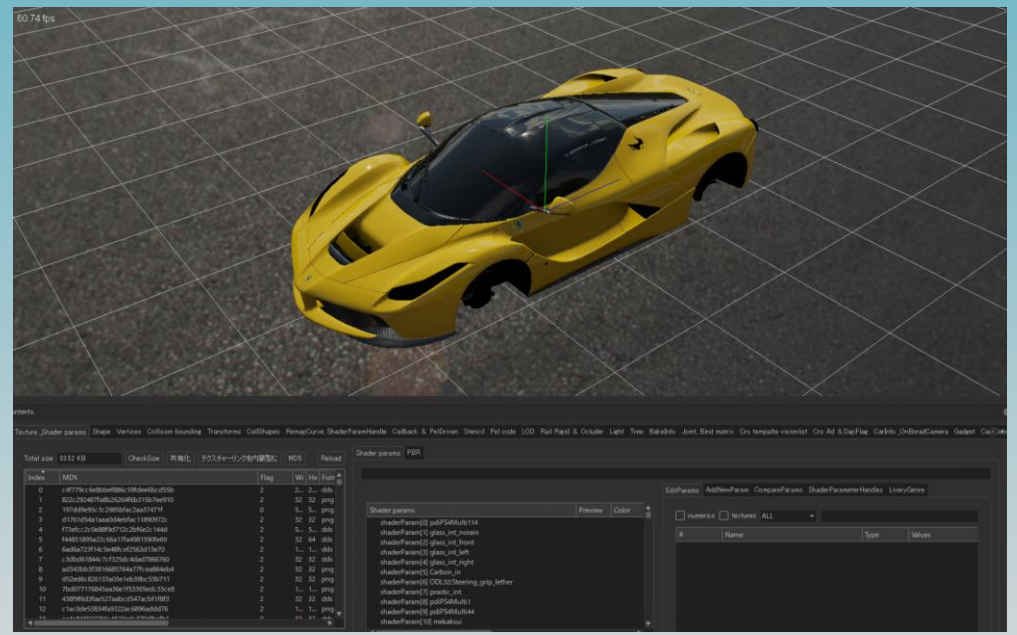


ポリフォニー・デジタルでのPython活用事例

- アセットのコンバーター
  - コンバーターもPythonで
  - アセットの最適化
  - ゲームデータの埋め込み
  - コンバートは、コンバートサーバーで実行される
  - コンバートエラーが発生したときに、ログに残るスタックトレースのおかげで不具合の解決が迅速

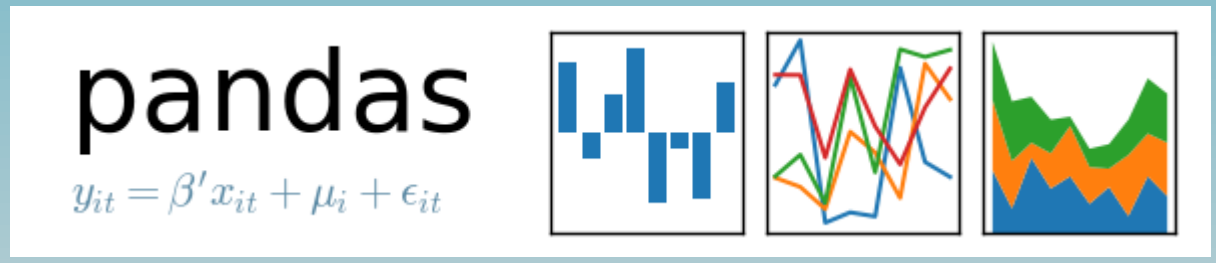
ポリフォニー・デジタルでのPython活用事例

- アセットデータのデバッグツール
  - ツールの機能はすべてPythonで実装
  - QWidget上でDirectX11描画
  - PythonからDirectX11のAPI呼び出し可能
  - アセットデータの描画は実機相当のシェーダー
- コンバート結果の調査に使用
  - シェーダーの設定
  - ライトの設定
  - データの最適化結果
  - 様々なデータの不具合発見

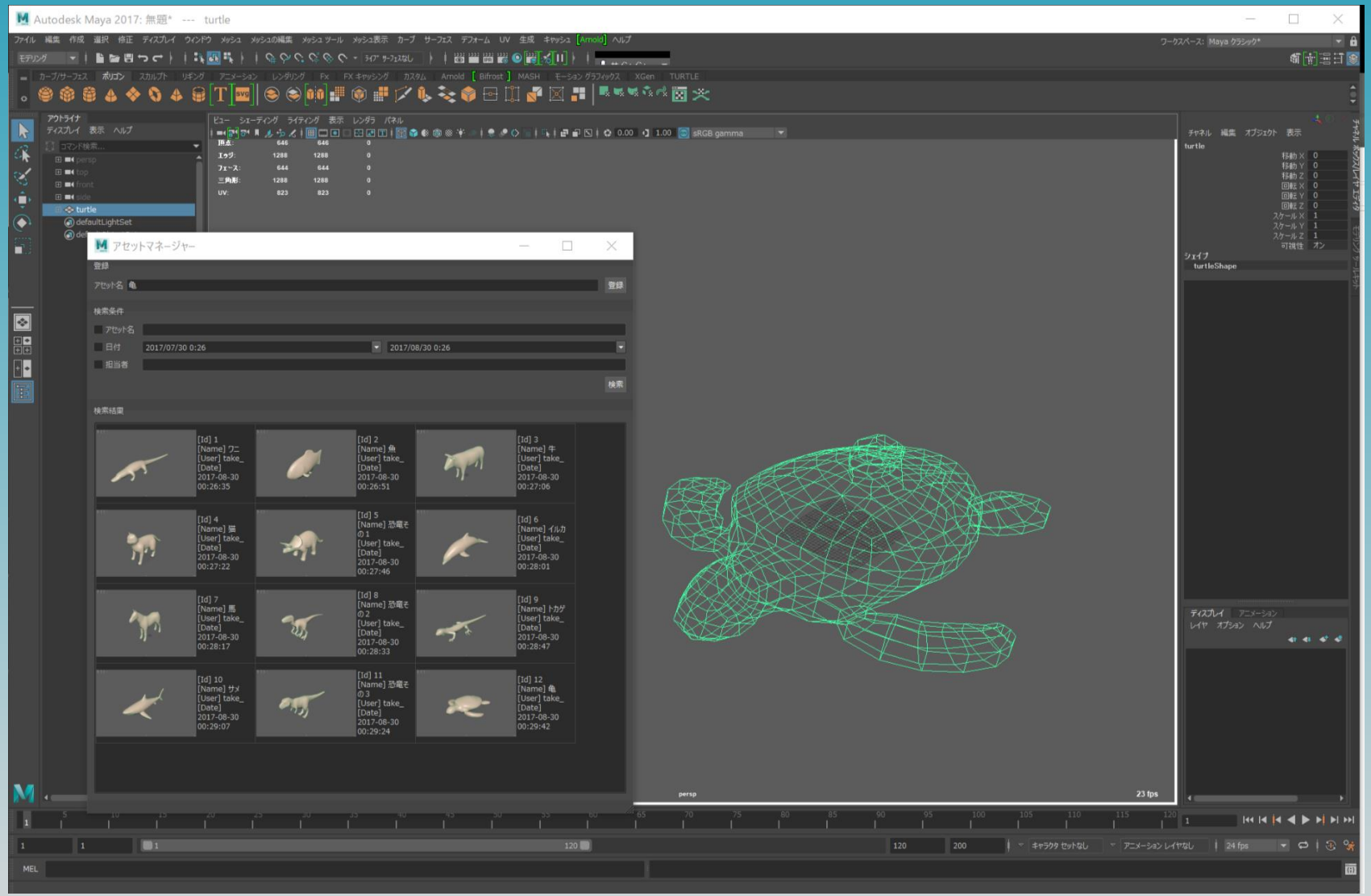


ポリフォニー・デジタルでのPython活用事例

- アセットデータの解析
  - Jupyter + pandas
  - 時系列でアセットのデータサイズの変化を可視化
  - アーティストのミスによる不自然なデータサイズの変更を検出



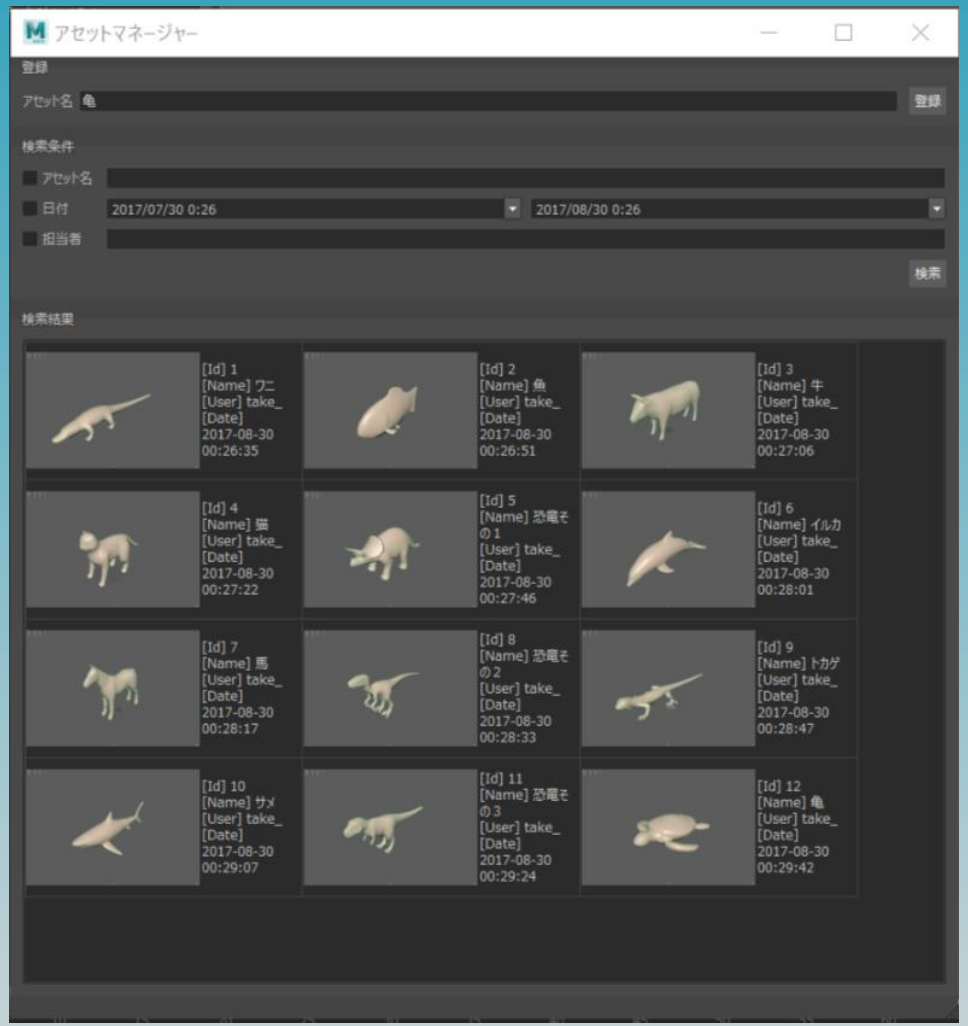
インハウスツールを作ってみよう！



今回の講演で使用したソースコードは下記から取得できます

https://github.com/takeno1027/cedec2017

アセット管理ツール


アセットマネージャー
登録
アセット名 亀
登録
検索条件
アセット名
日付 2017/07/30 0:26 2017/08/30 0:26
担当者
検索
検索結果
[Id] 1 [Name] ワニ [User] take_ [Date] 2017-08-30 00:26:35
[Id] 2 [Name] 魚 [User] take_ [Date] 2017-08-30 00:26:51
[Id] 3 [Name] 牛 [User] take_ [Date] 2017-08-30 00:27:06
[Id] 4 [Name] 猫 [User] take_ [Date] 2017-08-30 00:27:22
[Id] 5 [Name] 恐竜その1 [User] take_ [Date] 2017-08-30 00:27:46
[Id] 6 [Name] イルカ [User] take_ [Date] 2017-08-30 00:28:01
[Id] 7 [Name] 馬 [User] take_ [Date] 2017-08-30 00:28:17
[Id] 8 [Name] 恐竜その2 [User] take_ [Date] 2017-08-30 00:28:33
[Id] 9 [Name] トカゲ [User] take_ [Date] 2017-08-30 00:28:47
[Id] 10 [Name] サメ [User] take_ [Date] 2017-08-30 00:29:07
[Id] 11 [Name] 恐竜その3 [User] take_ [Date] 2017-08-30 00:29:24
[Id] 12 [Name] 亀 [User] take_ [Date] 2017-08-30 00:29:42

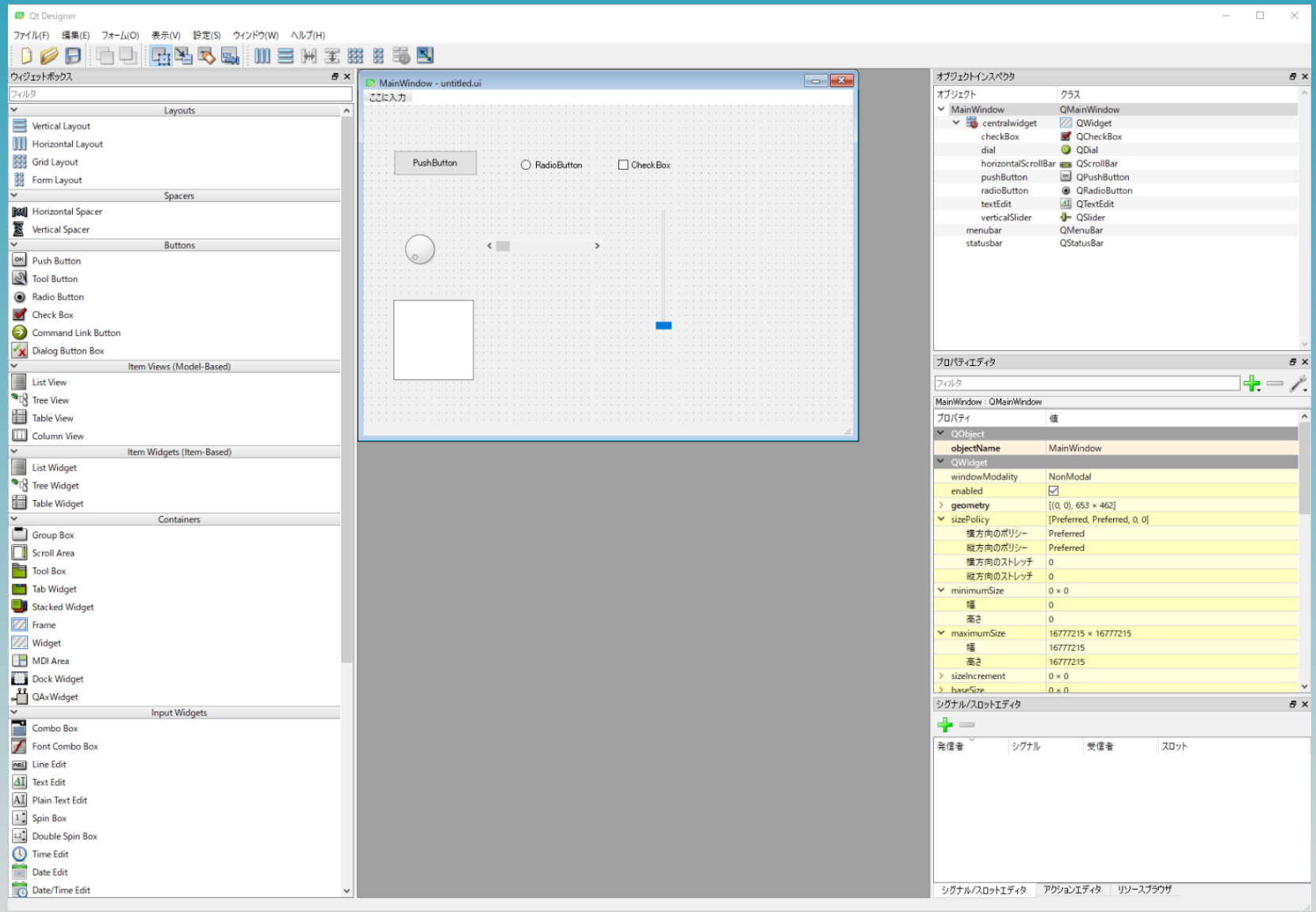
実装に必要な物

- お気に入りのテキストエディター(Spyder)
- Maya
- PySide2
- Qt Designer
- sqlite3
- SQLAlchemy

PySide2

- QtのPythonバインディング
- 同種にPyQt
- PySideとPyQt ライセンス形態の違い
- 機能に違いは“ほぼ”無い

Qt Designer


Qt Designer
ファイル(F) 編集(E) フォーム(O) 表示(V) 設定(S) ウィンドウ(W) ヘルプ(H)
ウィジェットボックス
フィルタ
Layouts
Vertical Layout
Horizontal Layout
Grid Layout
Form Layout
Spacers
Horizontal Spacer
Vertical Spacer
Buttons
Push Button
Tool Button
Radio Button
Check Box
Command Link Button
Dialog Button Box
Item Views (Model-Based)
List View
Tree View
Table View
Column View
Item Widgets (Item-Based)
List Widget
Tree Widget
Table Widget
Containers
Group Box
Scroll Area
Tool Box
Tab Widget
Stacked Widget
Frame
Widget
MDI Area
Dock Widget
QAxWidget
Input Widgets
Combo Box
Font Combo Box
Line Edit
Text Edit
Plain Text Edit
Spin Box
Double Spin Box
Time Edit
Date Edit
Date/Time Edit
MainWindow - untitled.ui
ここに入力
PushButton
RadioButton
CheckBox
オブジェクトインスペクタ
オブジェクト
クラス
MainWindow
QMainWindow
centralwidget
QWidget
checkBox
QCheckBox
dial
QDial
horizontalScrollBar
QScrollBar
pushButton
QPushButton
radioButton
QRadioButton
textEdit
QTextEdit
verticalSlider
QSlider
menubar
QMenuBar
statusbar
QStatusBar
プロパティエディタ
フィルタ
MainWindow : QMainWindow
プロパティ
値
QObject
objectName
MainWindow
QWidget
windowModality
NonModal
enabled
geometry
[(0, 0), 653 × 462]
sizePolicy
[Preferred, Preferred, 0, 0]
横方向のポリシー
Preferred
縦方向のポリシー
Preferred
横方向のストレッチ
0
縦方向のストレッチ
0
minimumSize
0 × 0
幅
0
高さ
0
maximumSize
16777215 × 16777215
幅
16777215
高さ
16777215
sizeIncrement
0 × 0
baseSize
0 × 0
シグナル/スロットエディタ
発信者
シグナル
受信者
スロット
シグナル/スロットエディタ
アクションエディタ
リソースブラウザ

sqlite3　SQLAlchemy

- sqllite3
  - ファイルベースの軽量DBエンジン
  - Pythonの標準モジュールに含まれる
- SQLAlchemy
  - 最も有名なPython用のORM

モックアップを作ってみよう

紙にUIデザインをスケッチ

UI要素の配置を試行錯誤

Qt Designerが便利！

アセット管理ツールに必要な要素

登録名

登録日時

登録実行ボタン

検索条件

検索実行ボタン

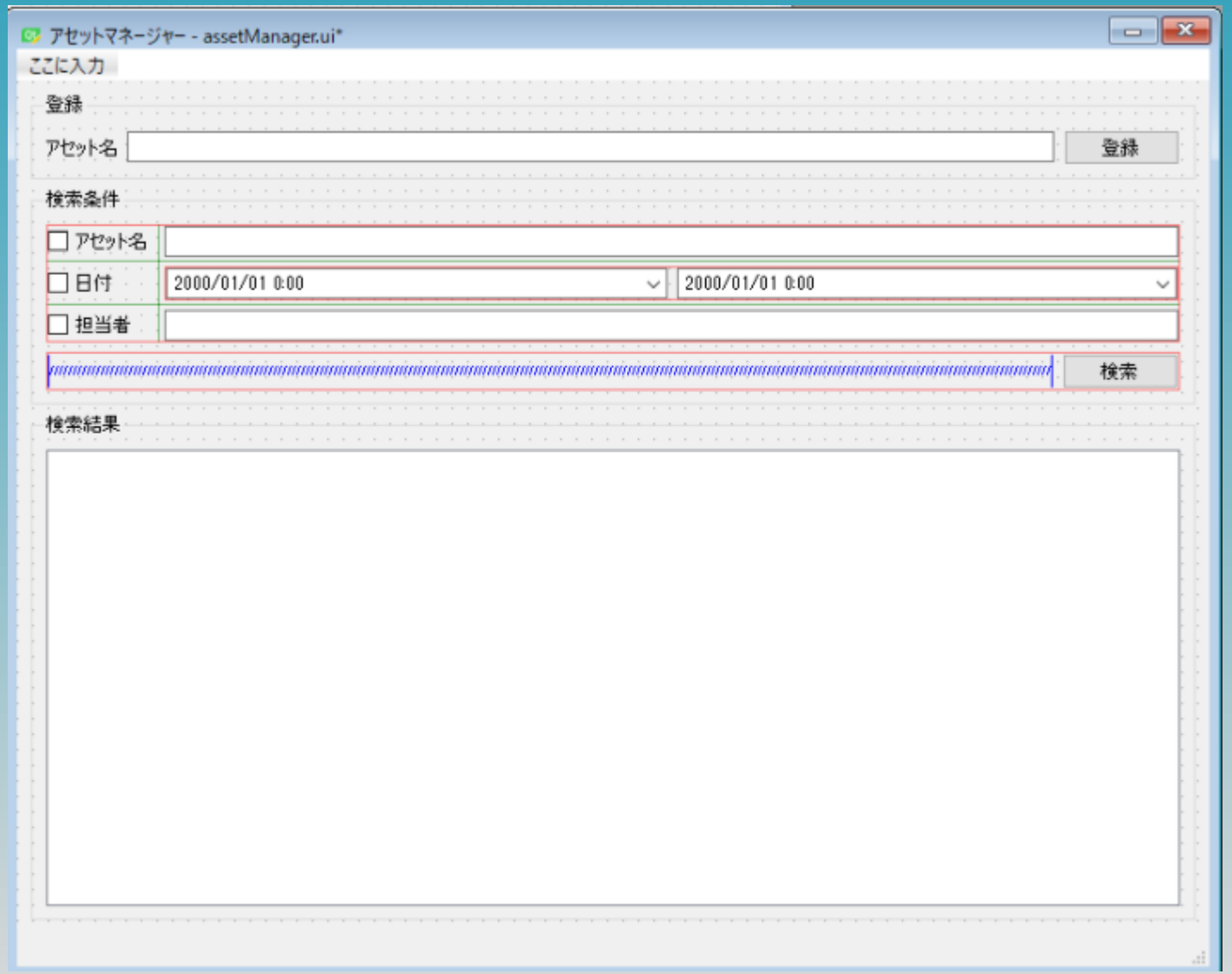
検索結果表示

Qt Designerでデザイン

登録欄

検索欄

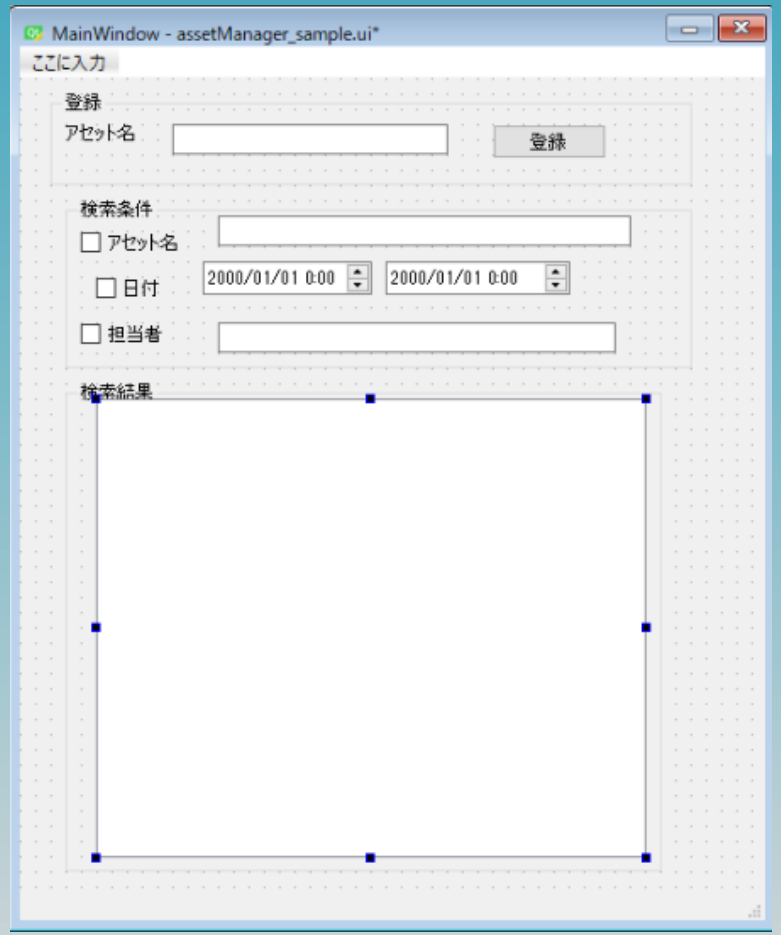
検索結果欄



3段で構成

実演

レイアウト



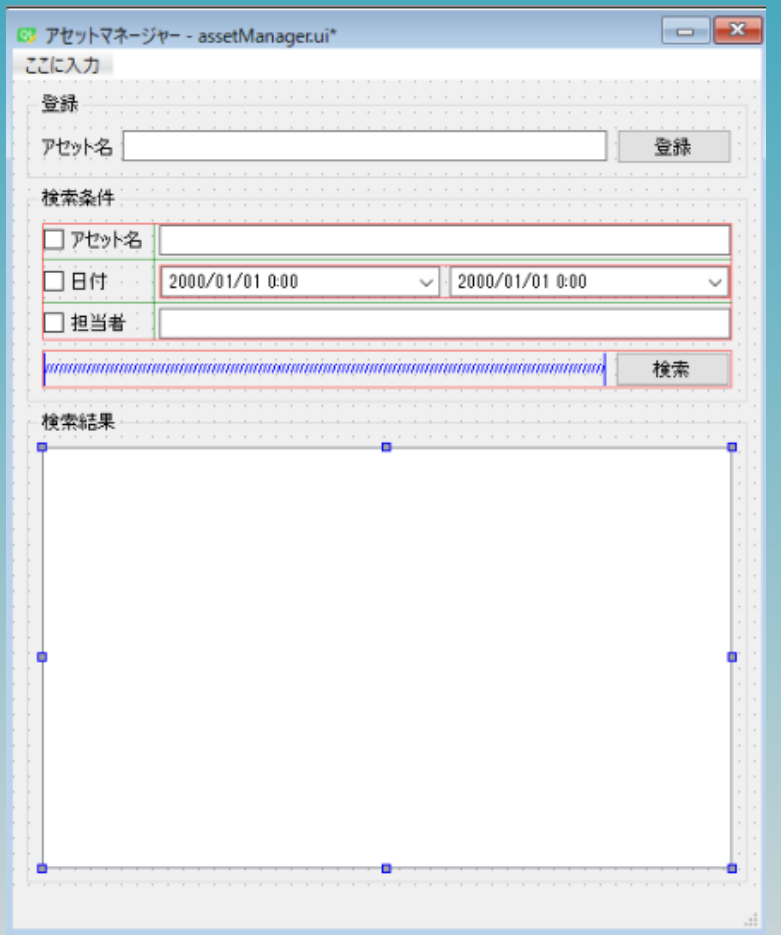
レイアウト機能を使う

水平レイアウト

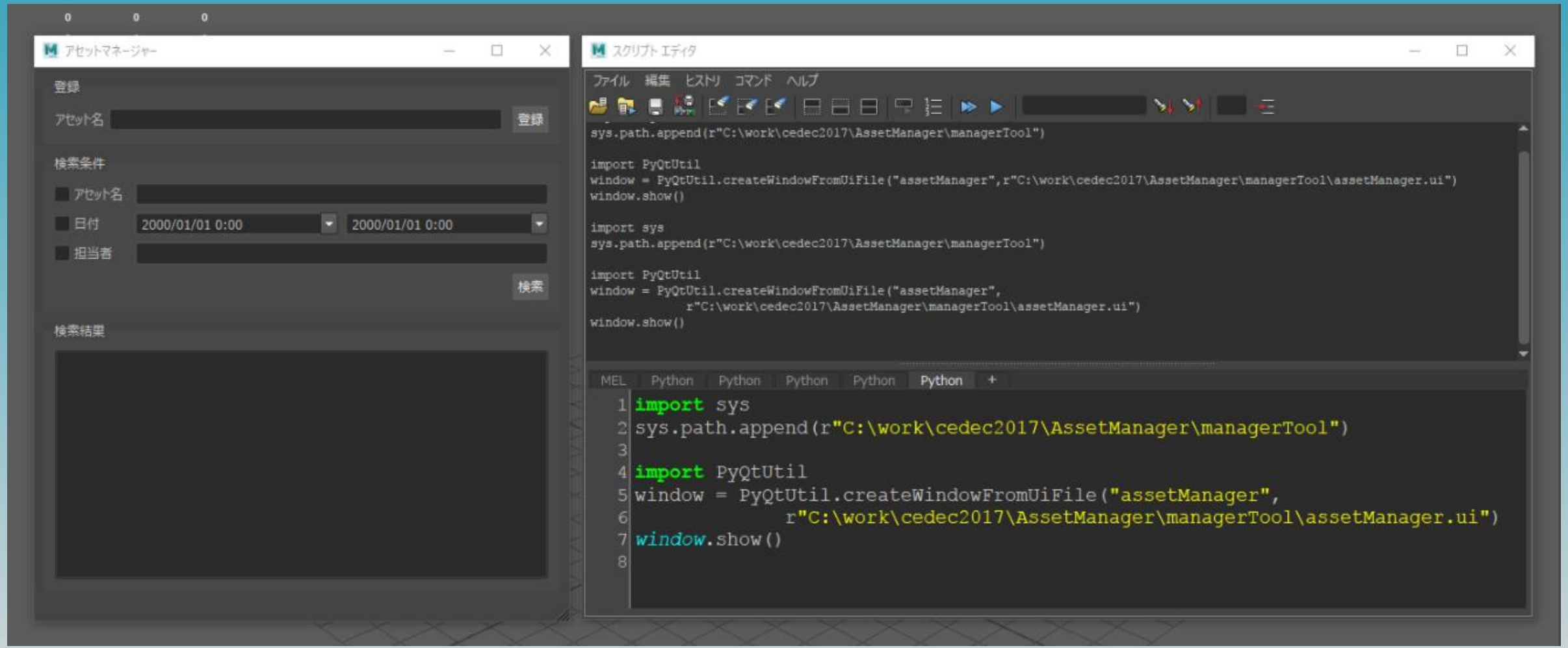
垂直レイアウト

グリッドレイアウト

ウィンドウをリサイズしても崩れない



モックアップをMaya上で表示してみよう



実演

PyQtUtil

- https://github.com/takeno1027/cedec2017 で公開
- PySide2,PyQt5の違いを吸収
  - shiboken2,sip
  - QUiLoader ,uic.loadUI
- UiファイルをロードしてQWidgetインスタンス生成
- アプリケーション固有情報を起動時復元、終了時永続化
- MayaのMMessageイベントをQtのシグナルとして扱える
  - シーンファイルがオープン
  - ノードがシーンに追加、削除

PyQtUtil の使い方

```
import sys
import os

from PyQtUtil import QtCore, QtWidgets, QtGui, createWindowFromUiFile, getMayaMainWindow

class AssetManagerWindow(QtWidgets.QMainWindow):
    def init_ui(self,appSettingData):
        pass

    def close_ui(self,appSettingData):
        pass

def main():
    window = createWindowFromUiFile('AssetManager',
                                    os.path.join(os.path.dirname(__file__),'assetManager.ui'),
                                    AssetManagerWindow(parent = getMayaMainWindow()))
    window.show()
```

Qtの特徴を理解する

- シグナル・スロット
- Model - View

シグナル･スロット

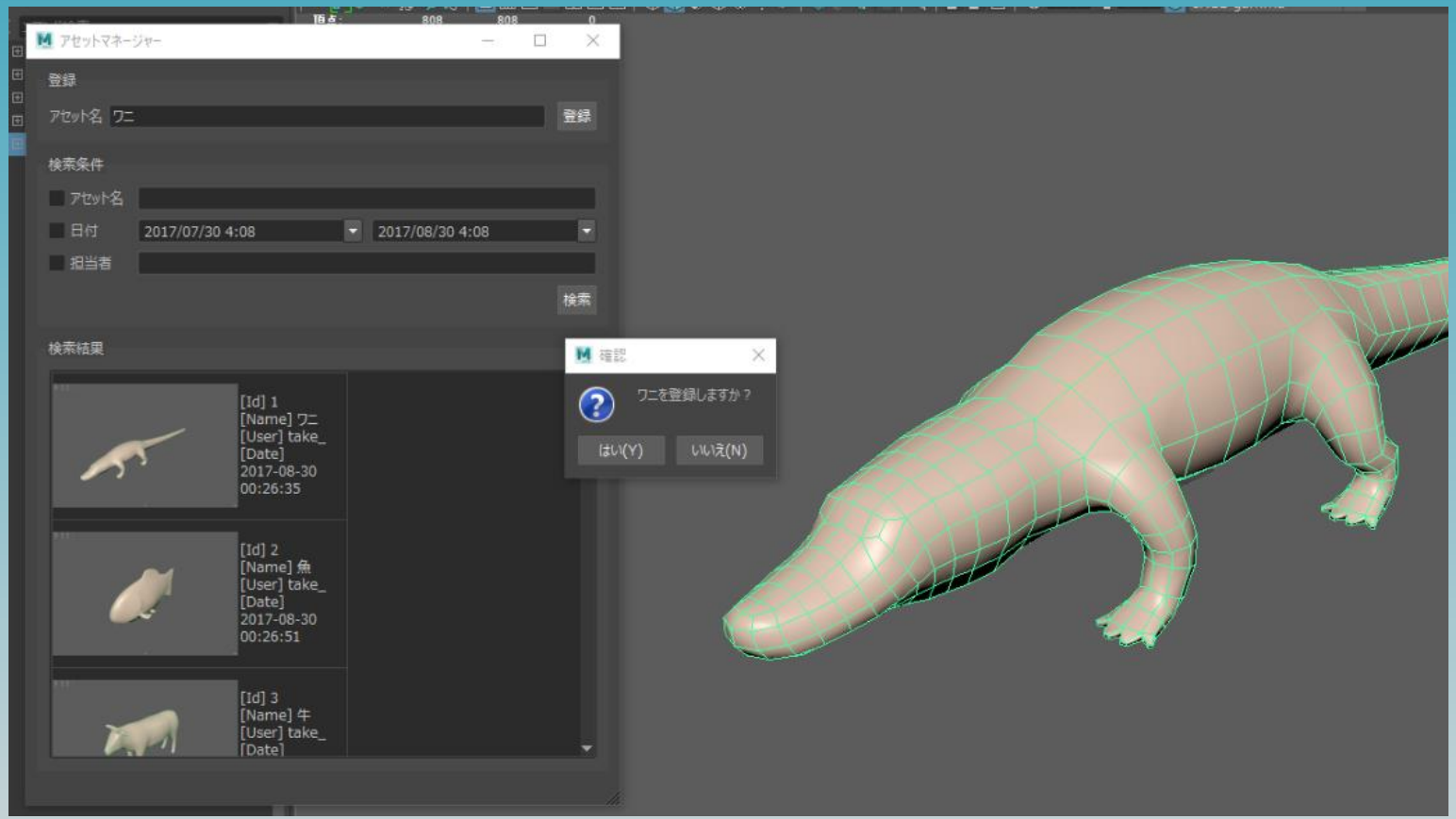
- ウィジットの状態の変更通知
  - ボタンが押された、数値入力、GUIで発生するイベント

```
class AssetManagerWindow(QtWidgets.QMainWindow):
    def init_ui(self,appSettingData):
        self.pushButton_insertDB.clicked.connect(self.insertDB)


    def insertDB(self):
           ...
```

Model-View

- Model　　Viewに表示させたいデータを管理する
- View　　Modelで管理したデータを表示する
- Qtには、MovelViewベース、Itemベースの2種類のWidget
  - QListView,QTreeView,QTableView,…
  - QListWidget,QTreeWidget,QTableWidget,…
- ModelViewのメリット
  - 同じデータ（Model)でViewを差し替えるとで、List,Tree,Tableと表示方法の入れ替えが容易

アセット登録の機能を実装する



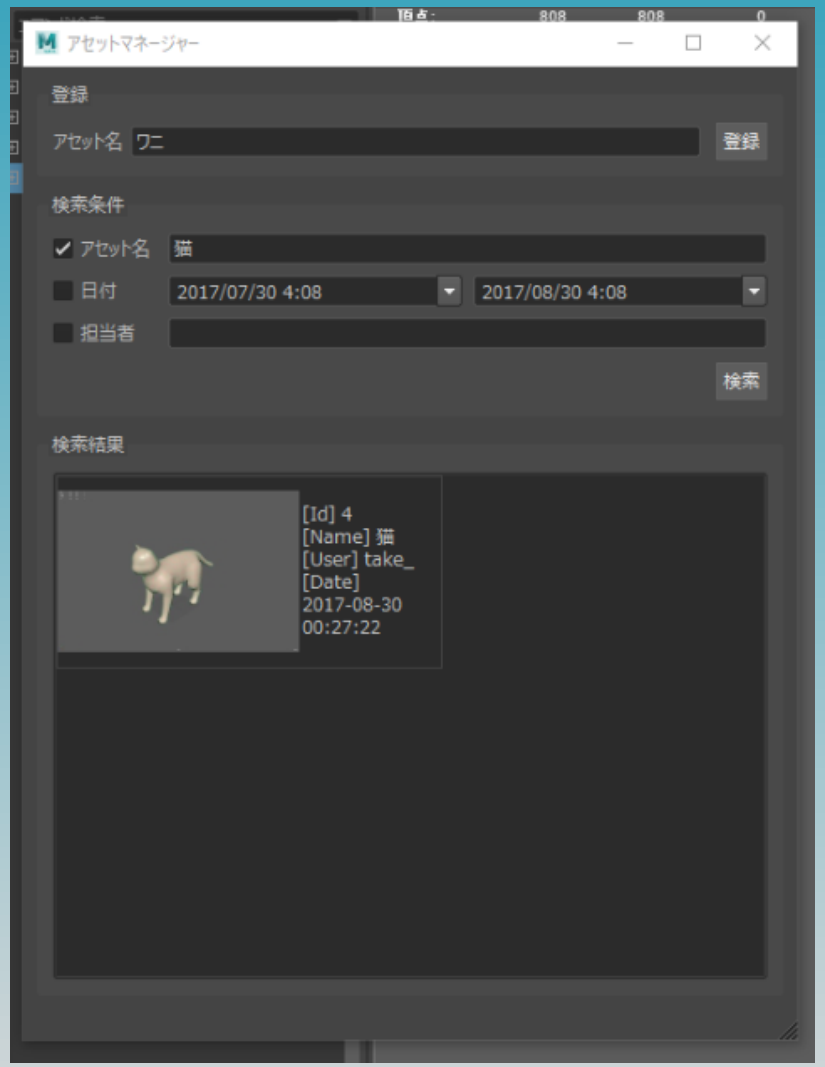
登録フロー

- 登録したいオブジェクトを選択
- アセット名を入力
- 登録ボタンを押す
- ボタンウィジットからシグナル発信
- 確認ダイアログ
- DBにinsert
- 登録IDがDBから返ってくる
- 登録IDのフォルダを生成
- シーンファイル、テクスチャーが登録IDのフォルダにコピー
- スナップショットを自動撮影してサムネイル画像も登録IDのフォルダに保存

アセット登録の機能を実装する

```
class AssetManagerWindow(QtWidgets.QMainWindow):
    def init_ui(self,appSettingData):
        self.pushButton_insertDB.clicked.connect(self.insertDB)
        self.connect = xmlrpclib.ServerProxy("http://localhost:15600")

    def insertDB(self):
        assetName = self.lineEdit_assetName.text()
        status = QtWidgets.QMessageBox.question(self,u'確認',u'%sを登録しますか？'  %
                                assetName,QtWidgets.QMessageBox.Yes | QtWidgets.QMessageBox.No)
        if status != QtWidgets.QMessageBox.Yes:
            return

        if len(cmds.ls(sl = True)) == 0:
            QtWidgets.QMessageBox.critical(self,u'エラー',u'登録したいオブジェクトを選択してください')
            return

        Id = self.connect.insertAsset(assetName,os.getenv('USERNAME'))
        print('Id : %d' % Id)
        exportedDirName = os.path.join(ASSET_ROOT_DIR,'%d' % Id)
        if not os.path.exists(exportedDirName):
            os.mkdir(exportedDirName)

        self.exportScene(exportedDirName)
        self.takeSnapShot(exportedDirName)
```

アセット検索の機能を実装する



検索フロー

- 有効にする検索条件をチェック
- 条件入力
- 検索ボタンを押す
- ボタンウィジットからシグナル発信
- DBに問い合わせ
- 検索結果がDBから返ってくる
- 検索結果からItemModelを更新
- このItemModelがセットされたTableViewに結果が表示

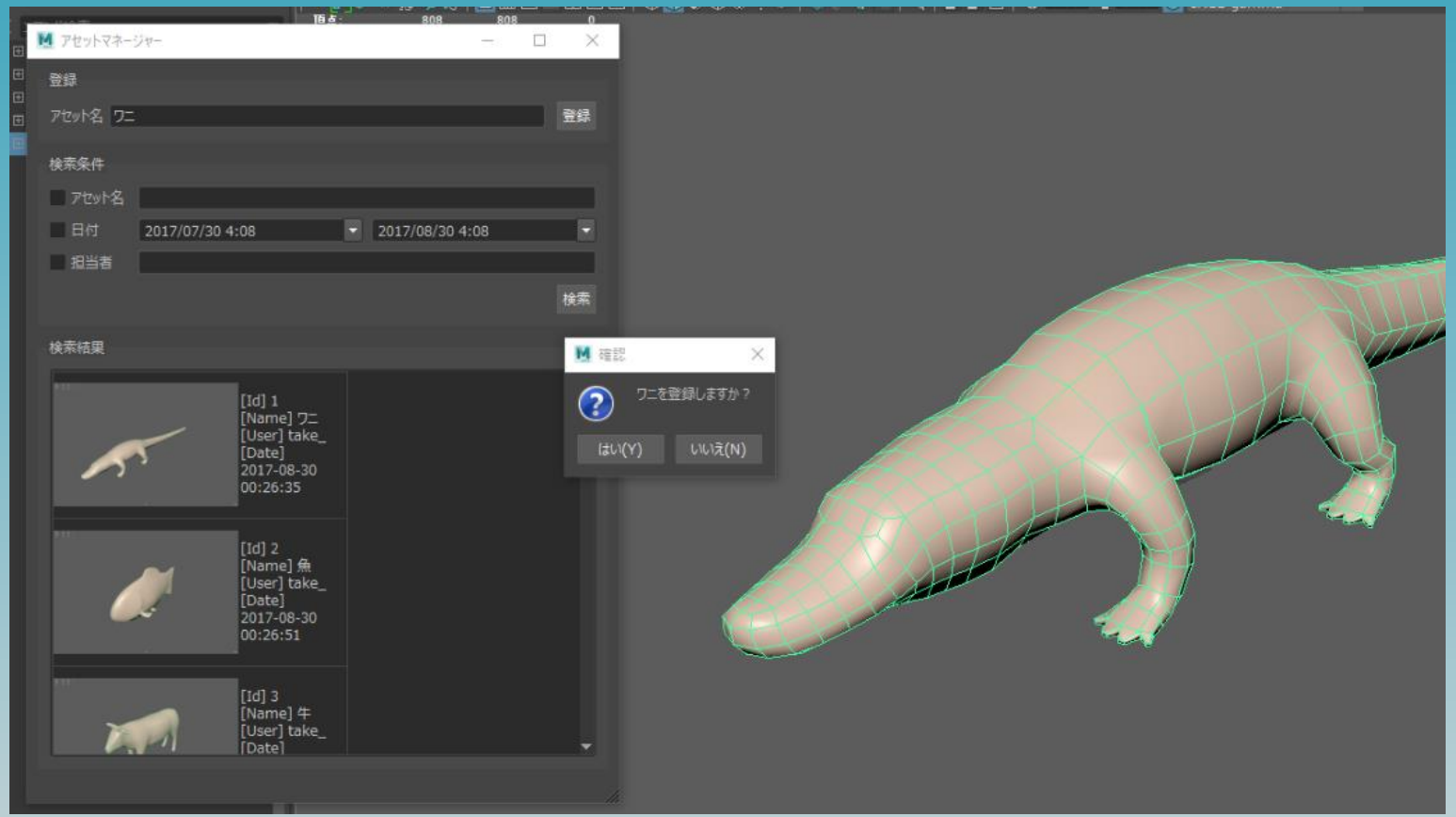
アセット検索の機能を実装する

```
class AssetManagerWindow(QtWidgets.QMainWindow):
    def init_ui(self,appSettingData):
        self.pushButton_searchDB.clicked.connect(self.searchDB)

        self.model = QtGui.QStandardItemModel()
        self.tableView_searchResult.setModel(self.model)

    def searchDB(self):
        conditions = {}

        if self.checkBox_assetName.checkState():
            conditions['name'] = self.lineEdit_search_assetName.text()

        self.searchResults = self.connect.searchAsset(conditions)
        self.updateSearchResult()


    def updateSearchResult(self):
         ...
```

Mayaとデータベースの接続

- DBエンジンは、sqlite3を今回は使用
- サーバー機能が無い
- 代わりにスタンドアローンで起動されるPythonでサーバーを建てる
- 標準モジュールのSimpleXMLRPC Serverを使用
- Maya側はアセット管理ツールの中で、XMLRPCクライアントを起動し、サーバーと通信

自動サムネイル生成


アセットマネージャー
登録
アセット名 ワニ
登録
検索条件
アセット名
日付
2017/07/30 4:08
2017/08/30 4:08
担当者
検索
検索結果
[Id] 1
[Name] ワニ
[User] take_
[Date]
2017-08-30
00:26:35
[Id] 2
[Name] 魚
[User] take_
[Date]
2017-08-30
00:26:51
[Id] 3
[Name] 牛
[User] take_
[Date]
確認
ワニを登録しますか？
はい(Y)
いいえ(N)

Widgetの中身をキャプチャ

- Qtには、Widgetの中身をキャプチャする関数が用意さている
  - QPixmap QWidget::grab(const QRect &rectangle)
- が、Mayaのビューポートを含むWidgetでgrab関数を呼び出しても、ビューポートの中身は、取り出せない。（真っ黒）

- 古典的な方法だが、glReadPixelsでOpenGLのフレームバッファを直接読み出す。
- ctypesを使ってOpenGL32.dllのglReadPixelsのエントリポイントを取得し、PythonからglReadPixelsを呼び出してみる

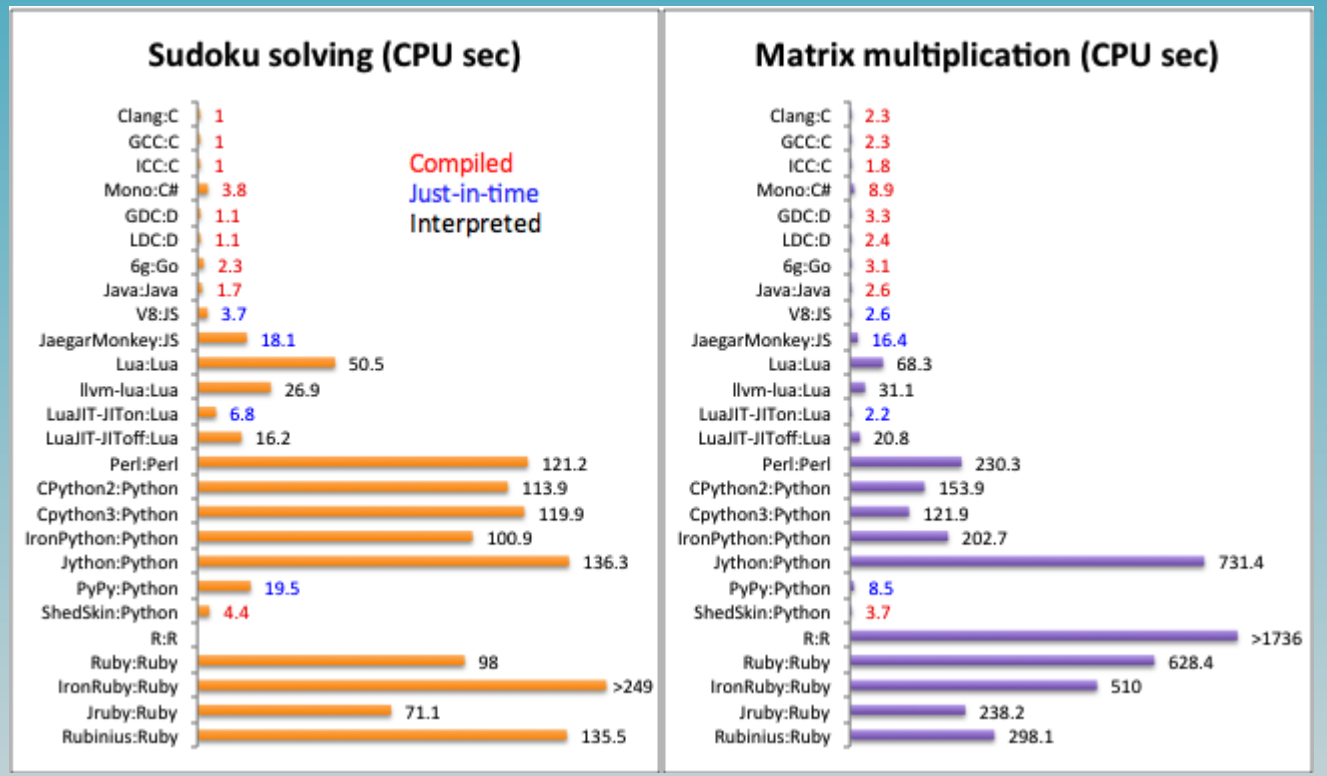
OpenGLの関数をPythonから呼び出す

```
import ctypes
glFT = ctypes.cdll.OpenGL32

glFT.glReadBuffer.argtypes = [ctypes.c_uint] # mode
glFT.glReadPixels.argtypes = [ctypes.c_int,ctypes.c_int, # x,y
                              ctypes.c_int,ctypes.c_int, # width, height
                              ctypes.c_uint,ctypes.c_uint, # format, type
                              ctypes.c_void_p] # data

def capture(snapShotImageFileName):
    view = OpenMayaUI.M3dView.active3dView()

    w = view.portWidth()
    h = view.portHeight()

    view.refresh()
    view.beginGL()

    img = OpenMaya.MImage()
    img.create(w,h)

    glFT.glReadBuffer(GL_FRONT)
    glFT.glReadPixels(0,0,w,h,GL_RGBA,GL_UNSIGNED_BYTE,ctypes.c_void_p(img.pixels()))

    view.endGL()

    ext = os.path.splitext(snapShotImageFileName)[1][1:]
    img.writeToFile(snapShotImageFileName,ext)
```

実演

Python高速化Tips

- 他の言語と比べて、遅い



[Programming Languages Benchmarks - GitHub Pages]

プロファイルの方法

- ボトルネック発見にはプロファイリングが必要

- timeモジュールによる時間計測
- cProfileモジュールによる関数ごとの呼び出し時間の計測
- line_profilerモジュールによるソースコードの行単位での計測

timeモジュールによる時間計測

```
import time

tm = time.time()
somefunc()
print("%f sec" % (time.time() - tm))
```

関数ごとの呼び出し時間計測

```
import cProfile
import pstats

profile = cProfile.Profile()
profile.enable()

somefunc()

profile.disable()
ps = pstats.Stats(profile)
ps.sort_stats("time")
ps.print_stats()
```

関数ごとの呼び出し時間計測

```
         22089029 function calls (4452389 primitive calls) in 6.712 seconds

   Ordered by: internal time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
17649648/13008    5.434    0.000    5.633    0.000 averageVertexColor.py:19(traverseAdjacencyListRecursive)
        1    1.077    1.077    6.712    6.712 averageVertexColor.py:121(averageTraverseAdjacencyListRecursive)
  4426368    0.199    0.000    0.199    0.000 {method 'append' of 'list' objects}
    13009    0.001    0.000    0.001    0.000 {len}
        1    0.000    0.000    0.000    0.000 {range}
        1    0.000    0.000    6.712    6.712 averageVertexColor.py:103(average)
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
```

ソースコードの行単位での計測

- line_profilerを使う
  - スタンドアローンのPython環境
    - pip install line_profiler (http://**aka.ms/vcpython27**が先にインストール必要)
  - Anaconda環境
    - conda install line_profiler

計測したい関数に@profileデコレーターを
つける

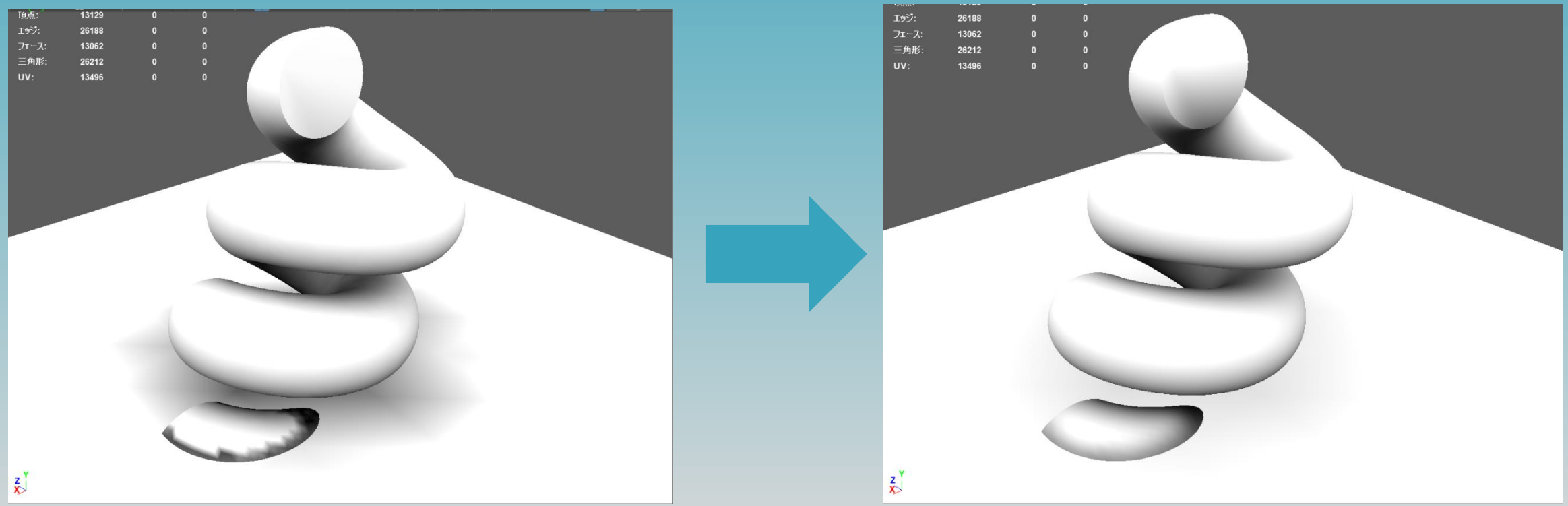
```
@profile
def somefun():

kernprof -l -v somefunc.py
```

ソースコードの行単位での計測

```
Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
   124                                           @profile
   125                                           def averageTraverseAdjacencyListRecursive(self):
   126         1            7      7.0      0.0      resultVertexColors = []
   127
   128         1            6      6.0      0.0      numVertexColors = len(self.vertexColors)
   129
   130     13009        15847      1.2      0.0      for i in range(0,numVertexColors):
   131     13008        40116      3.1      0.1          accumulate = []
   132     13008     41030616   3154.3     61.1          self.traverseAdjacencyListRecursive(i,0,accumulate)
   133
   134     13008        19682      1.5      0.0          color = [0.0,
   135     13008        12672      1.0      0.0                   0.0,
   136     13008        12432      1.0      0.0                   0.0,
   137     13008        24442      1.9      0.0                   self.vertexColors[i][3]]
   138
   139   4426368      4281577      1.0      6.4          for j in accumulate:
   140   4413360      4932753      1.1      7.3              c = self.vertexColors[j]
   141
   142   4413360      5523649      1.3      8.2              color[0] += c[0]
   143   4413360      5587240      1.3      8.3              color[1] += c[1]
   144   4413360      5630199      1.3      8.4              color[2] += c[2]
   145
   146     13008        19312      1.5      0.0          cnt = len(accumulate)
   147
   148     13008        18894      1.5      0.0          color[0] /= cnt
   149     13008        16981      1.3      0.0          color[1] /= cnt
   150     13008        17001      1.3      0.0          color[2] /= cnt
   151
   152     13008        20468      1.6      0.0          resultVertexColors.append(color)
   153
   154         1            4      4.0      0.0      self.vertexColors = resultVertexColors
```

高速化実例

- 頂点カラー平均化ツール
  - 焼かれたAOを馴染ませるなど



アルゴリズムの説明

- MItMeshVertexイテレータで各頂点の隣接リストを構築する
- ある深さまで隣接リストをトラバースし、その頂点カラーの総和を平均化する
- とても単純なアルゴリズム

```
vertItr = OpenMaya.MItMeshVertex(shape)

while not vertItr.isDone():
    vertexColors.append(list(vertItr.getColor()))
    connectedVertices.append(list(vertItr.getConnectedVertices()))

    vertItr.next()

averageVertexColor(vertexColors,connectedVertices)
```

再帰を使った実装

```
def traverseAdjacencyListRecursive(self,i,depth,accumulate):
        """
        隣接リストの走査　再帰
        """
        if depth == self.maxDepth:
            return

        accumulate.append(i)

        for j in self.connectedVertices[i]:
            self.traverseAdjacencyListRecursive(j,depth+1,accumulate)
```

再帰をループに変更した実装

```
def traverseAdjacencyListLoop(self,start):
        """
        隣接リストの走査　ループ化
        """
        accumulate = [[start]]
        for i in range(0,self.maxDepth-1):
            tmp = []
            for j in accumulate[i]:
                for k in self.connectedVertices[j]:
                    tmp.append(k)

            accumulate.append(tmp)

        return list(chain.from_iterable(accumulate))
```

再帰処理をループに置き換えると、高速化される。
Pythonの関数呼び出しのコストが高いことがわかる

ループをリスト内包表記に変更した実装

```
def traverseAdjacencyListLoop2(self,start):
        """
        隣接リストの走査 内ループをリスト内包表記
        """

        accumulate = [[start]]

        for i in range(0,self.maxDepth-1):
            accumulate.append([k for j in accumulate[i] for k in self.connectedVertices[j]])

        return list(chain.from_iterable(accumulate))
```

内ループをリスト内包表記に変更すると、生成されるバイトコードが減ることで高速化される

ローカル変数化に変更した実装

```
def traverseAdjacencyListLoop3(self,start):
        """
        隣接リストの走査 内ループをリスト内包記法
        dotアクセス（辞書の検索を避ける）
        """

        accumulate = [[start]]
        append  = accumulate.append
        maxDepth = self.maxDepth
        connectedVertices = self.connectedVertices

        for i in range(0,maxDepth-1):
            append([k for j in accumulate[i] for k in connectedVertices[j]])

        return list(chain.from_iterable(accumulate))
```

頻繁にアクセスされるメンバ変数、メンバ関数をローカルに定義し直すと、速くなる

まとめ

- ポリフォニー・デジタルでは、多くのアセット制作ツールをPython + PySideで実装しています。
- Qt Designerを使えば、かっこいいUIのツールが簡単に作れます。
- Pythonを高速化する手段は、他にもあります。
  - 今回、ご紹介できませんでしたが、cythonを使ったネイティブコードへの変換といったテクニックもあります。
- ツール、パイプライン開発に興味を持つ開発者がたくさん増えると嬉しいです。

質問は？

今回の講演で使用したソースコードは下記から取得できます

https://github.com/takeno1027/cedec2017